Etude de la désintégration du Boson Z0

Philippe Laurent

LABORATOIRE DE PHYSIQUE CORPUSCULAIRE

Rapport de stage Janus

Etude de la désintégration du Boson Z^0

Laurent Pierre

Directeur de stage : M. Philippe Gris

Sommaire

1. Introduction

Dès mon arrivée à l'Université Blaise Pascal, je me suis intéressé au métier de chercheur. Lorsque j'ai découvert la possibilité de réaliser un stage au sein du Laboratoire de Physique Corpusculaire de Clermont-Ferrand, j'ai immédiatement proposé ma candidature. Ce stage a été pour moi une formidable occasion de découvrir les méthodes de travail d'un chercheur, l'organisation d'un laboratoire ainsi que les sujets d'étude et les outils employés dans la recherche.

Le but de mon stage Janus était double ; une initiation au monde de la recherche, mais aussi la découverte de domaines de la physique que j'avais peu ou pas étudiés (la relativité restreinte [1] et la physique des particules [2]), afin de pouvoir travailler sur la désintégration du boson Z^0. J'ai aussi développé mes connaissances en informatique, jusque là limité à la programmation en langage C, en découvrant la programmation orientée objet en langage C++ [3], puis l'utilisation d'un outil très puissant et très utile aux chercheurs : les librairies Root.

Mon rapport se partagera en trois axes principaux. En premier lieu, je rappellerai les nouvelles bases théoriques que j'ai acquises, des équations de Boost de Lorentz jusqu'au Boson Z^0, un des médiateurs de l'interaction nucléaire faible, produit au Tevatron, à Fermilab. La description de ce collisionneur de particules et des détecteurs qui lui sont associés sera le thème de ma seconde partie. Je décrirai enfin l'ensemble de mes travaux et productions personnelles effectués lors du stage et conclurai par une comparaison entre mes résultats et ceux mesurés par les scientifiques.

2. Un peu de théorie

2.1. Relativité Restreinte et Boost de Lorentz

La Relativité Restreinte est une théorie élaborée en 1905 par Albert Einstein, reposant sur deux postulats :

- Les lois de la physique ont la même forme dans tous les référentiels inertiels (ou galiléens) ;
- La vitesse de la lumière dans le vide a la même valeur dans tous les référentiels inertiels.

Cette théorie introduit à l'époque une idée révolutionnaire dans le monde de la physique : contrairement aux phénomènes décrits par la mécanique newtonienne, la Relativité Restreinte décrit un cadre physique dans lequel les métriques spatiales et temporelles ne sont pas invariantes, mais dépendent du mouvement de l'observateur par rapport au phénomène. Cela se traduit par les formules de Lorentz, qui permettent d'exprimer les coordonnées (x,y,z,t) d'un évènement dans un référentiel fixe en fonction de celle d'un référentiel en mouvement à la vitesse v suivant l'axe Oz (x',y',z',t') (c représentant la vitesse de la lumière) :

$$\begin{cases} ct = \gamma(ct' + \beta x') \\ z = \gamma(z' + \beta ct') \\ y = y' \\ x = x' \end{cases}$$

$$avec\ \beta = v/c\ et\ \gamma = \frac{1}{\sqrt{1-\beta^2}}$$

Afin d'obtenir une quantité invariante, il est alors nécessaire d'établir une métrique d'espace-temps : l'intervalle d'espace-temps entre deux évènements est alors invariante par transformation de Lorentz :

$$s_{12} = c^2(t_1 - t_2)^2 - ((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)$$

Introduisons maintenant les formules de Lorentz dans le cadre de la physique des particules. On s'intéresse cette fois à une particule définie par le quadrivecteur (P_x,P_y,P_z,E) dans son référentiel propre et par le quadrivecteur (P_x',P_y',P_z',E') dans le référentiel du laboratoire (on parle alors de Boost de Lorentz). On utilise la convention $c = \hbar = 1$ (le boost est encore réalisé suivant Oz) :

$$\begin{cases} E' = \gamma(E + \beta P_z) \\ P_x' = P_x \\ P_y' = P_y \\ P_z' = \gamma(P_z + \beta E) \end{cases}$$

2.2. Le Modèle Standard de physique des particules

Le Modèle Standard (MS) (*Figure 1*) est une théorie modélisant le comportement des particules physiques ainsi que les quatre interactions fondamentales. Il est basé sur l'union des théories de la mécanique quantique et de la relativité restreinte : on parle alors de théorie quantique des champs. Suivant ce modèle, les particules de matière, nommées fermions (de spin 1/2, au nombre de douze, six saveurs de quarks, six saveurs de leptons) interagissent entre elles par échange d'autres particules, nommées bosons (une pour chaque interaction fondamentale, de spin 1). Par exemple, lorsque deux électrons se repoussent, cette interaction est interprétée comme l'échange d'un photon entre les deux particules.

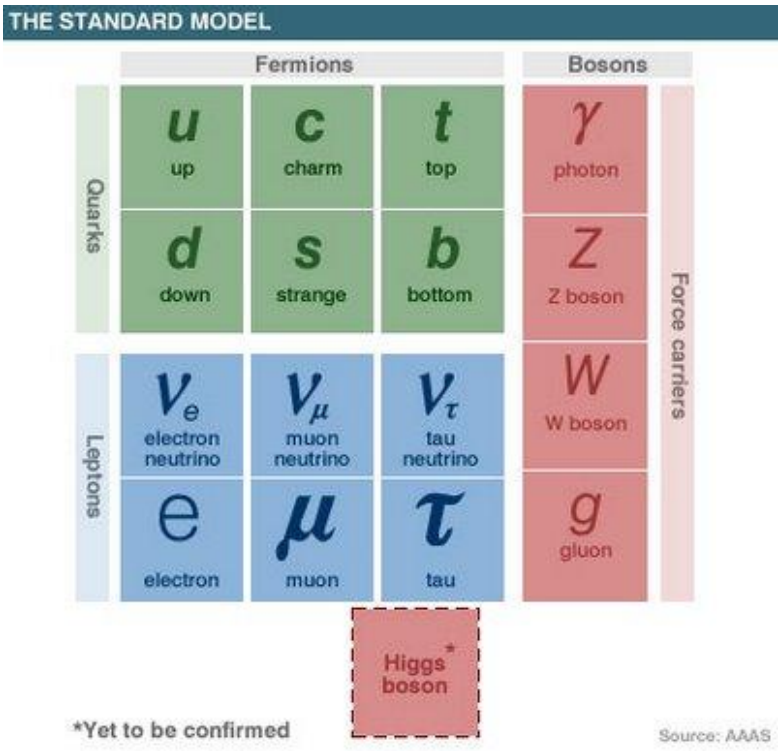

Figure 1 : Particules du Modèle Standard

Cette théorie, développée de 1970 à 1973, a été bien vérifiée expérimentalement : les masses des particules observées dans les accélérateurs viennent en effet appuyer la véracité de ce modèle. Un problème persiste cependant : l'existence de l'hypothétique Boson de Higgs, une particule

intervenant dans le mécanisme qui donne une masse aux autres particules Il n'a toujours pas été observé et est nécessaire à la validation complète de la théorie.

2.3. Les diagrammes de Feynman

Pour représenter les interactions entre particules, on utilise communément un code très pratique : le diagramme de Feynman. Dans cette représentation, l'ordonnée représente le temps, l'abscisse le mouvement de la particule (sans pour autant montrer ni direction ni vitesse). Une ligne représente une particule (elle peut prendre diverses formes en fonction de la particule représentée : une ligne droite pour un fermion, ondulée pour un photon), le croisement de 3 lignes un vertex. Si l'on reprend l'exemple de nos électrons, on obtient simplement le diagramme suivant :

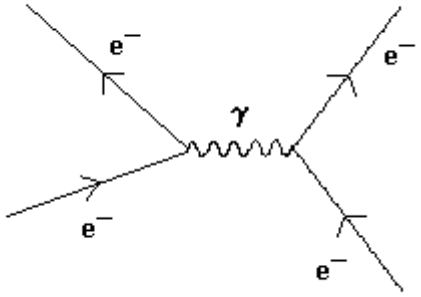

Figure 2 : Interaction entre deux électrons

2.4. Les bosons $W^{\pm}$ et Z^0

La force nucléaire faible est à l'origine de la désintégration nucléaire. Elle est portée selon le modèle standard par trois particules massives, les bosons $W^{\pm}$ (de masse 80,423 ± 0,039 GeV [4]) et le boson Z^0 (de masse 91,1876 ± 0,0021 GeV [4]). L'émission et l'absorption des bosons $W^{\pm}$ permettent aux quarks de changer de saveur, tandis qu'un boson Z^0 peut se désintégrer en une paire quark/anti-quark ou en une paire lepton/anti-lepton (de même saveur dans les deux cas). Les bosons $W^{\pm}$ sont responsables du fait que les quarks et les leptons les plus massifs se désintègrent rapidement en d'autres quarks et leptons, plus légers. Notons que le poids important de ces particules est la cause de la très faible portée de la force nucléaire faible.

Prenons un exemple : la désintégration d'un neutron en proton se réalise par l'émission d'un boson W^-, qui se désintègre par la suite en un électron et un antineutrino. Le diagramme de Feynman de cette réaction se représente de la manière suivante :

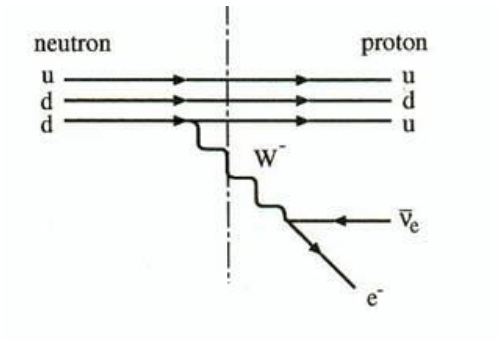

Figure 3 : Désintégration d'un neutron en proton

La principale réaction étudiée lors de ce stage sera la désintégration d'un boson Z^0 en deux muons (avec émission de photons), réalisée dans le collisionneur de particules à Fermilab, dans la banlieue de Chicago.

3. L'expérience D0

3.1. Le Tevatron

Le Tevatron (*Figure 4*) est le deuxième plus puissant collisionneur de particules dans le monde, derrière le LHC (Large Hadron Collider) de Genève, au CERN. Situé à Fermilab, dans la banlieue de Chicago, sa construction s'achève en 1983. Il génère des faisceaux de protons et d'antiprotons (on parle alors de collisionneur hadronique), qui sont accélérés à l'aide de puissants aimants supraconducteurs puis circulent en sens inverse dans l'anneau principal, d'un diamètre de 2 kilomètres (*Figure 5*). Un système de lentilles focalise les rayons en plusieurs points précis afin d'y réaliser des collisions proton/antiproton à une énergie de 1.96 TeV dans le centre de masse : c'est à ces endroits que sont placés les détecteurs CDF (Collider Detector) et D0, au sein desquels ont lieu près de 2,5 millions de collisions par seconde.

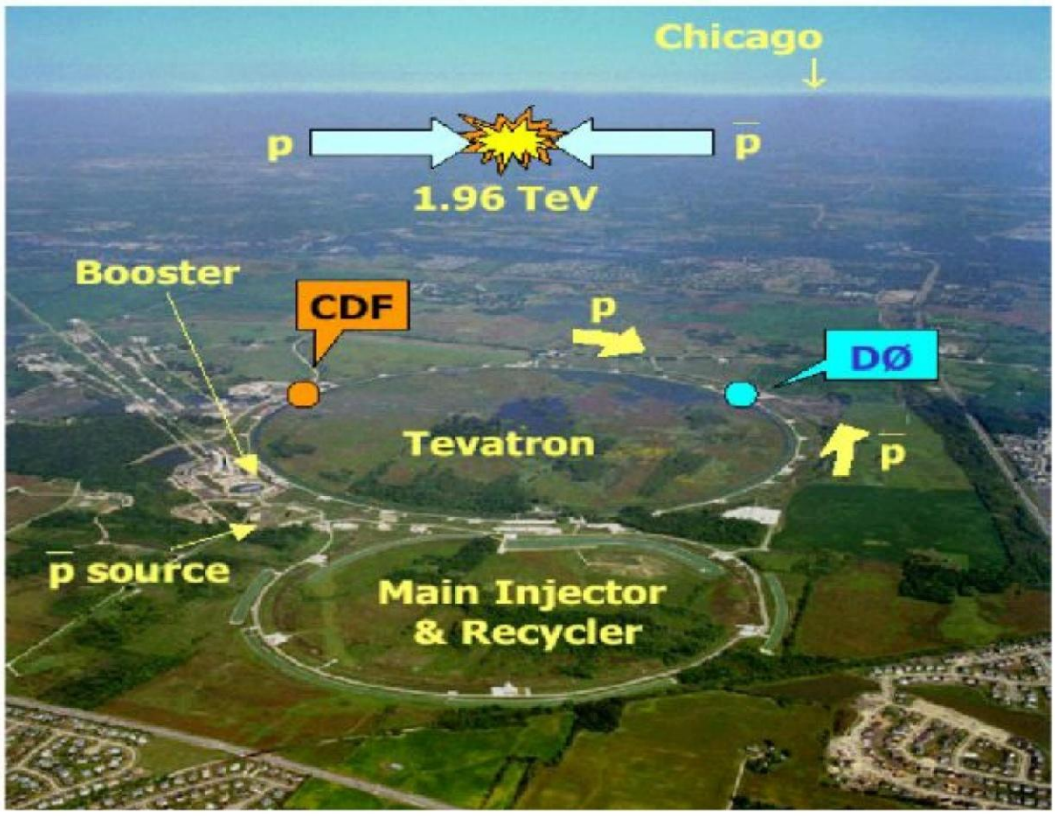

Figure 4 : Vue aérienne du Tevatron

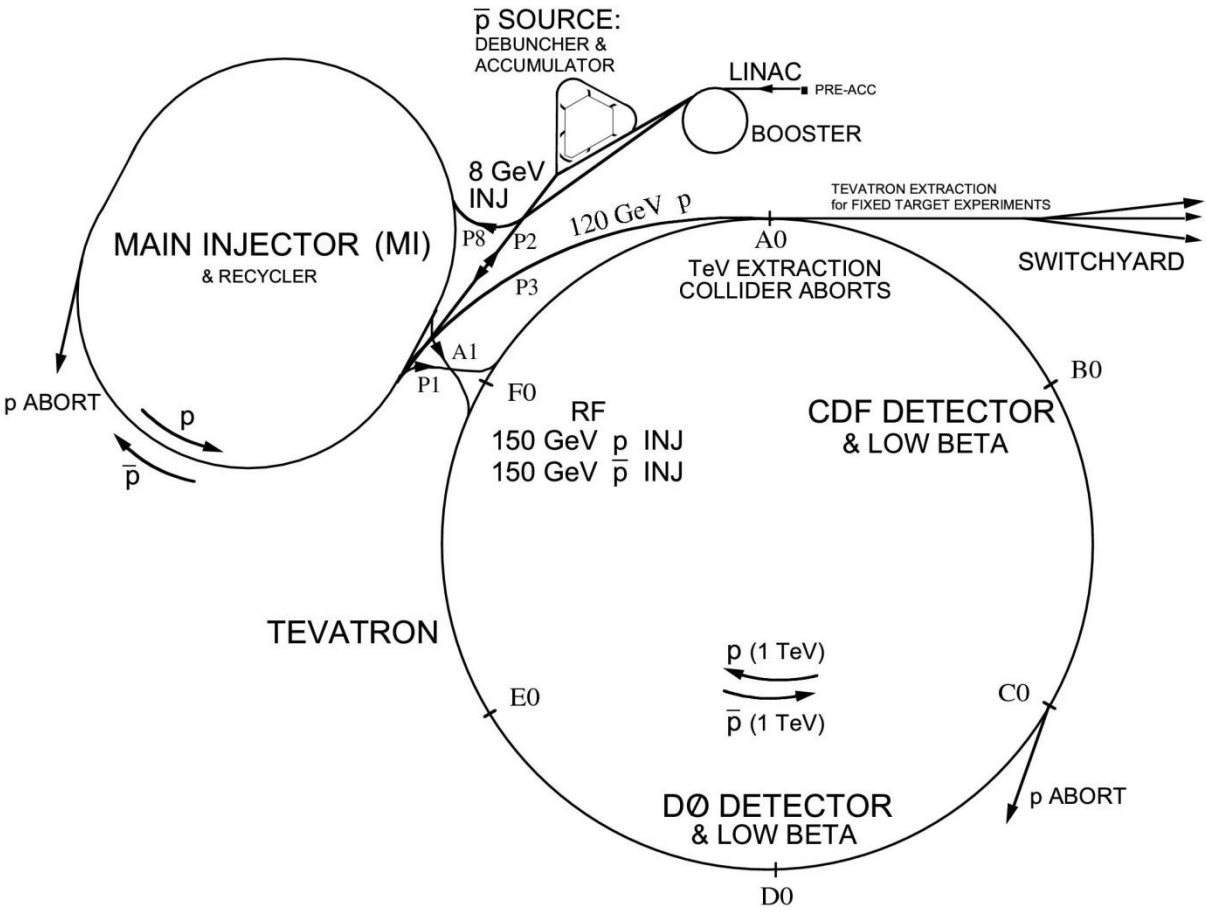

Figure 5 : Schéma du complexe du Tevatron

Les premières collisions ont été observées le 13 Octobre 1985. Le 2 Mars 1995, les chercheurs travaillant sur les détecteurs CDF et D0 annoncent la découverte du quark top, le dernier quark non découvert mais prédit par le Modèle Standard, qui contribue à appuyer davantage la véracité du modèle théorique. Sa masse a été mesurée avec précision (174,3 ± 5.1 GeV [4]) par les deux collaborations.

Limité aujourd'hui par son énergie (1,96 TeV face aux collisions de 14 TeV du LHC), le Tevatron devrait s'arrêter de fonctionner vers la fin de l'année 2011 (il aura fonctionné pendant environ 25 ans, durée de vie moyenne d'une expérience en physique des particules).

3.2. Les détecteurs de l'expérience D0

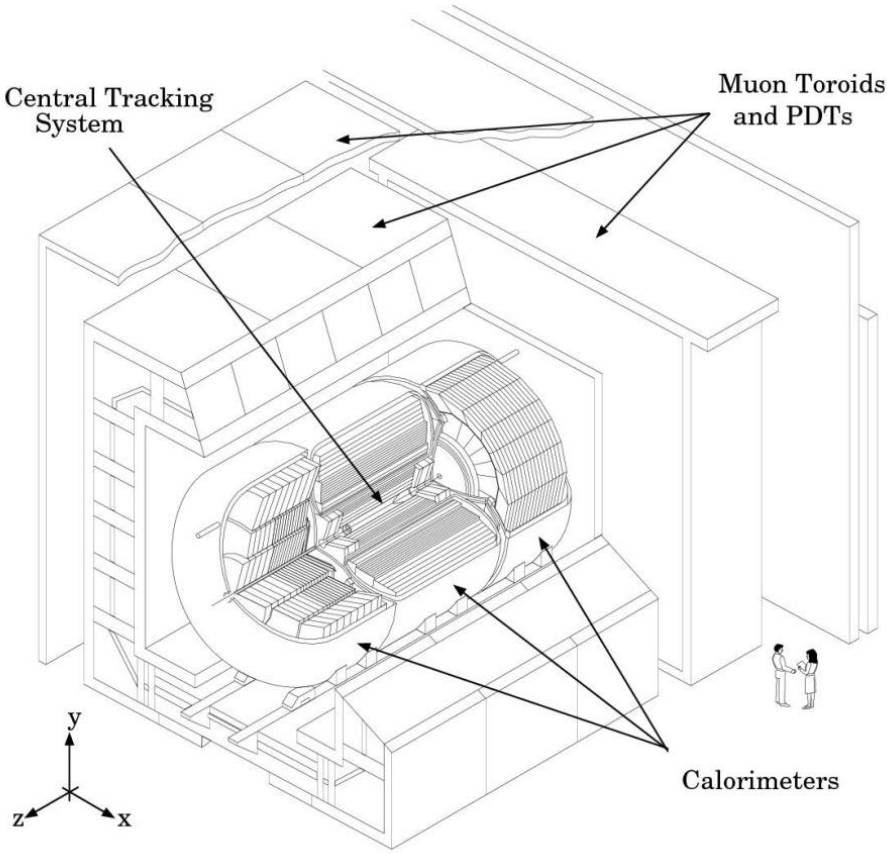

Figure 6 : Vue isométrique du détecteur D0

Le trajectographe et la chambre à muons du détecteur D0 (*Figure* 6 - *7*) baignent chacun dans un champ électromagnétique permettant de mesurer l'impulsion d'une particule. En effet, la trajectoire d'une particule est victime de multiples altérations lors de son passage à proximité des divers atomes situés au voisinage de sa trajectoire. Si la particule se déplace dans un champ magnétique uniforme d'intensité *B*, on peut connaître la valeur de son impulsion *P*, le rayon de courbure de sa trajectoire étant proportionnel à *P/B*. Il ne reste plus qu'à reconstruire la trajectoire de la particule et mesurer son énergie : elles sont mesurées à l'aide de plusieurs détecteurs, disposés en couches, du centre à la périphérie [5][6] :

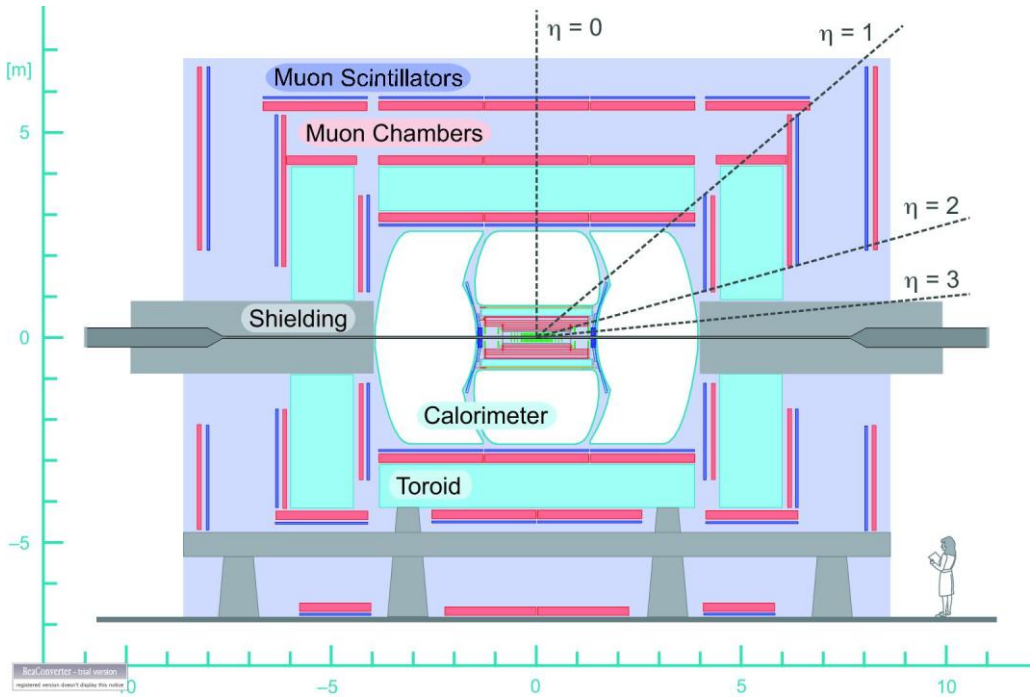

Figure 7 : Vue en coupe du détecteur D0

- *Trajectographe*

Le lieu de la collision est entouré de détecteurs qui enregistrent les trajectoires des particules générées par le choc proton/antiproton. Les relevés des trajectoires proches de la collision sont réalisés à l'aide de détecteurs en silicium. Couteux à réaliser, mais très précis (à la centaine de µm près), ils ne sont utilisés que sur une distance réduite autour des faisceaux de particules. En dehors du silicium sont utilisées des chambres à scintillation, composées de fibres qui émettent des photons lors du passage d'une particule.

- *Calorimètre*

Le calorimètre, situé à l'extérieur du détecteur de trajectoires, mesure l'énergie des particules le traversant. Il est composé d'uranium (ou de fer) baigné dans de l'argon liquide : les particules traversant le calorimètre réagissent avec l'uranium et perdent de l'énergie, tandis que l'argon permet la détection de ces réactions en donnant un signal électrique mesurable. On peut ainsi retrouver l'énergie des particules traversant le calorimètre.

- *Détecteur à muons*

La partie extérieure du détecteur permet de repérer les muons. Ces particules ont une durée de vie suffisante ($(290{,}6 \pm 1.1) \times 10^{-15}$ s [4]) pour s'échapper du détecteur, et ne sont pas absorbées par le calorimètre, mais elles sont importantes, car elles permettent de repérer les collisions les plus intéressantes.

- *Système de déclenchement*

Près de 2,5 millions de collisions par secondes ont lieu à l'intérieur du détecteur, alors que seulement 50 000 évènements par seconde peuvent être stockés sur les systèmes de sauvegarde informatisés. Le déclenchement est un système électronique et informatique qui détecte en temps réel si l'évènement produit est suffisamment intéressant pour être conservé.

- Efficacité / Pureté

Deux caractéristiques sont importantes pour estimer la validité de la reconnaissance d'un événement : l'efficacité et la pureté. L'efficacité représente la probabilité de reconnaître une particule produite dans le détecteur, la pureté représente la probabilité de dire que la particule observée *n'est pas* telle ou telle particule. Une bonne efficacité entraîne généralement une faible pureté (c'est-à-dire beaucoup d'évènements relevés, mais certainement beaucoup de déchets). L'inverse induit un faible nombre d'évènements, mais qui sont tous valides. La configuration choisie (beaucoup de pureté ou beaucoup d'efficacité) dépend de la réaction que l'on cherche à étudier : en effet, une faible efficacité sur un événement qui se produit rarement dans le collisionneur risque de conduire le physicien à ne pas observer suffisamment d'évènements pour en tirer des conclusions valables.

4. Mes productions

4.1. Découverte du langage C++ : la réalisation d'un boost de Lorentz simple suivant un axe

La première partie de mon stage a consisté à assimiler les bases du langage C++ (nécessaire pour la programmation des librairies Root) ainsi que les bases théoriques de la relativité restreinte. La première mise en pratique de ces connaissances a été la programmation d'un boost de Lorentz simple : une particule P de masse 90 GeV et d'énergie E dans le référentiel du laboratoire de 200 GeV qui se désintègre en deux autres particules ultrarelativistes P_1 et P_2 (on peut alors négliger leurs masses), avec des trajectoires back-to-back (angle égal à π entre les deux particules produites). Le programme calcule les facteurs β et γ à l'aide des paramètres d'entrées, puis booste les quadrivecteurs énergie-impulsion des particules P_1 et P_2 (voir annexe 1).

- *Dans le référentiel propre de la particule (ou référentiel du centre de masse des produits de désintégration) :*

$$M = E_1 + E_2 = E$$

$$\overrightarrow{P_1} = -\overrightarrow{P_2}$$

$$P_1 = P_2 = E_1 = E_2 = M / 2 = 45\ GeV$$

- *Dans le référentiel du laboratoire :*

$$\overrightarrow{P'} = \overrightarrow{P_1'} + \overrightarrow{P_2'}$$

$$E' = E_1' + E_2'$$

- *Calcul des facteurs β et γ en partant des équations du Boost de Lorentz :*

$$E' = \gamma(E + \beta P_z)$$

$$P_z{}' = \gamma(P_z + \beta E)$$

Dans le référentiel du centre du masse, les composantes de l'impulsion sont nulles, la particule est au repos, d'où :

$$E' = \gamma E$$

Puisque l'on traite le cas d'une particule ultra-relativiste (donc $E = M$) :

$$\gamma = E'/M$$

De même :

$$\beta\gamma = P_z'/E = P_z'/M$$

D'où

$$\beta = P_z'/E' = P'/E'$$

Les premiers résultats obtenus avec ce programme (en effectuant des tests simples directement sous la console Linux) semblent concluants. En effet, l'impulsion et l'énergie sont bien conservées. On constate aussi que l'angle α entre P_1 et P_2 tend vers 0 lorsque l'on augmente la valeur de l'énergie de la particule P dans le référentiel du laboratoire. Il s'agit maintenant de vérifier cela en utilisant les librairies Root [7].

4.2. Découverte et utilisation des librairies Root

Pour vérifier la véracité des résultats obtenus ci-dessus, j'ai créé un second programme en remplaçant mes classes par celle des librairies Root. Les résultats obtenus sont les mêmes que précédemment.

En faisant varier les données d'entrées (l'angle θ_1 entre les directions de $\overrightarrow{P_1}$ et de $\vec{P}$ et l'énergie de la particule P de 0 à 9000 GeV), on obtient toujours un angle α qui semble tendre vers 0. Cependant, en observant de plus près la répartition des valeurs de l'angle α, on remarque qu'il ne tend pas exactement vers 0 (*Figure 8*).

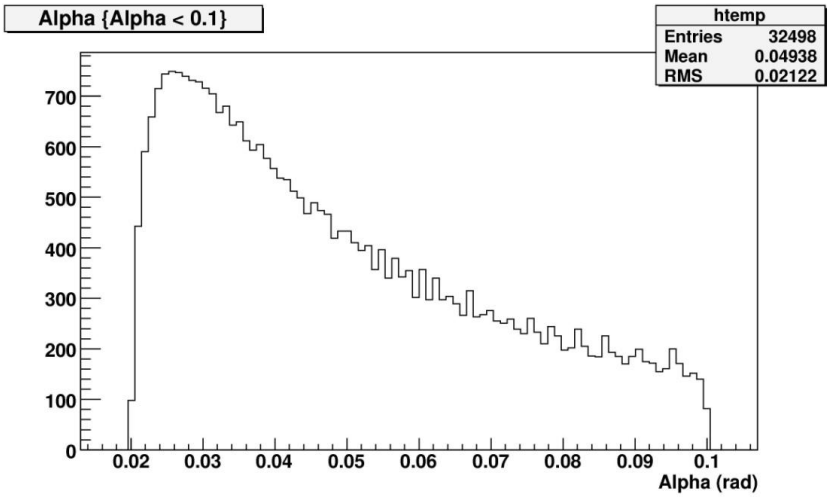

Figure 8 : Angle limite

Pour que cet angle soit nul, il faut que lorsque les directions des particules P_1 et P_2 sont confondues avec celle de l'axe de boost ($\theta_1 = 0$ et $\theta_2 = \pi$), le boost soit suffisamment important pour « retourner » la particule P_2. Cela n'arrive jamais dans le cas étudié (cas d'une particule ultra-relativiste). Calculons maintenant la valeur de l'angle α maximal, qui intervient logiquement quand $\theta'_1 = \pi / 2$ et $\theta'_2 = -\pi / 2$ dans le référentiel du centre de masse : on a alors Pz'_1 et Pz'_2, qui correspondent aux coordonnées de leurs impulsions suivant l'axe du boost, égales à 0.

$$P'_{z1} = \gamma(P_{z1} - \beta E_1) = 0 \rightarrow P_{z1} = \beta E_1$$

$$avec\ P_{z1} = P_1 * \cos\theta \rightarrow \cos\theta = (\beta E_1)/P_1$$

$$De\ plus\ P_1 = E_1 car M_1 = 0$$

$$donc\ \cos\theta = \beta$$

$$\theta = arcos(\beta)$$

$$donc\ \alpha = 2arcos(\beta)$$

La valeur de l'angle entre les deux particules est limitée par la valeur de β. On vérifie ce résultat pour la valeur maximale de l'énergie de P dans le référentiel du laboratoire (9000 GeV). On s'attend à retomber sur la valeur minimale de α :

$$\gamma = E/M = 9000/90 = 100$$

$$\beta = \sqrt{(1 - 1/\gamma^2)} = 0{,}9999$$

$$\alpha = 2arcos\beta = 2arcos(0{,}9999) = 0{,}02rad$$

On retrouve les résultats obtenus par le programme (en plus des traditionnelles conservations de l'énergie et de l'impulsion).

Pour finir, on réalise un boost sur l'ensemble de l'espace en faisant varier tous les paramètres en coordonnées sphériques, aussi bien pour la direction de l'axe du boost que la direction des particules P_1 et P_2 par rapport à l'axe du boost (voir annexe 2). Cette fois, on fait simplement varier l'énergie de la particule P de 200 à 800 GeV, par pas de 200 GeV. Les observations précédentes se vérifient à nouveau : plus l'énergie de boost est importante, plus l'angle α entre les deux particules est faible (*Figure 9*), et quand l'angle β' entre la direction de P1 et l'axe du boost vaut 0 (les directions sont confondues), l'angle α vaut toujours π (*Figure 10*).

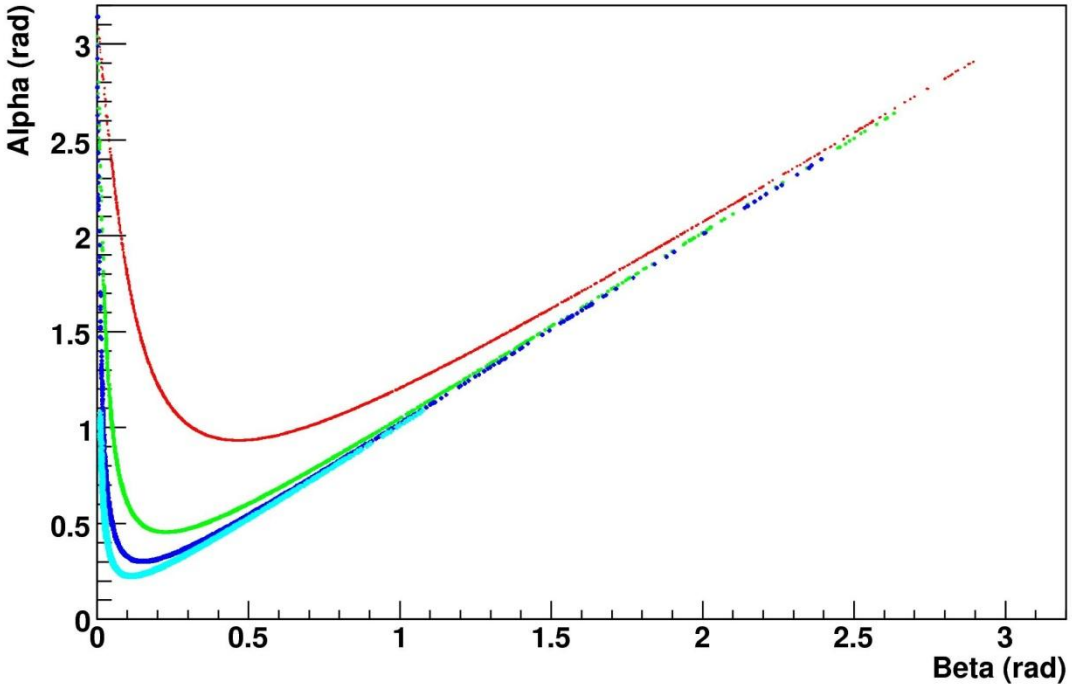

Figure 9 : Angle α en fonction de l'angle β (angle entre la direction du boost et l'axe des abscisses). Les courbes, de la plus fine à la plus épaisse, représentent respectivement les valeurs obtenues avec une énergie de Boost de 200 GeV, 400 GeV, 600 GeV et 800 GeV.

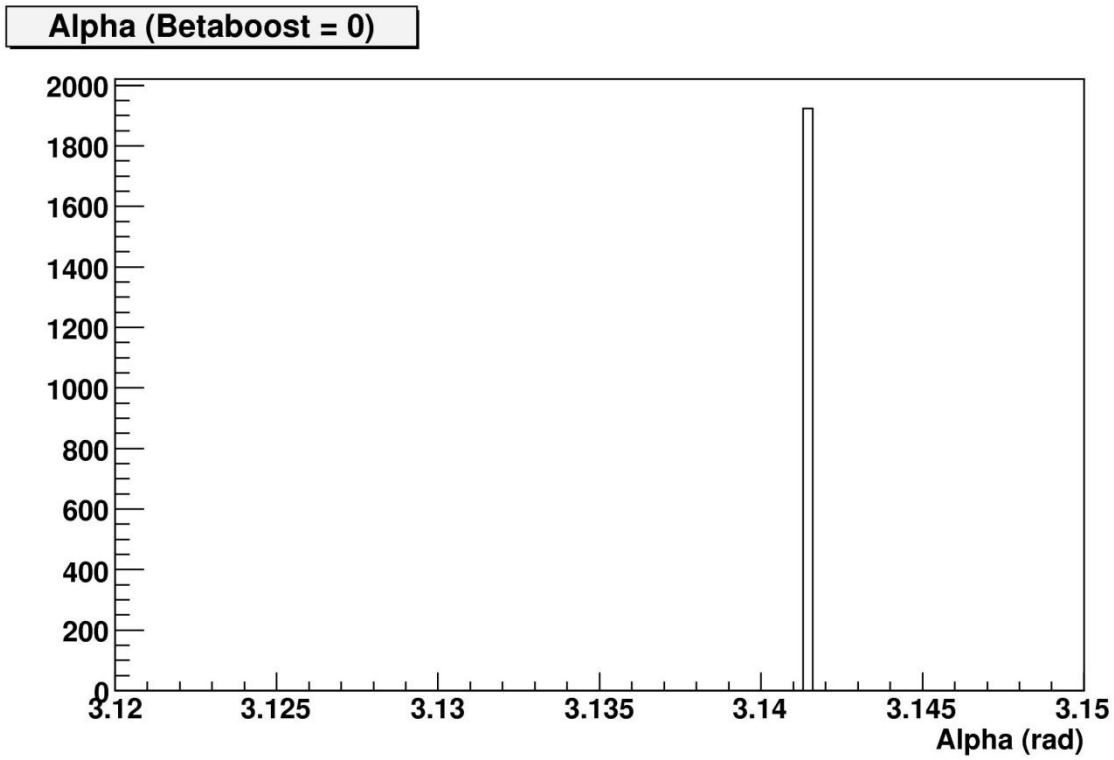

Figure 10 : Angle dans le référentiel du laboratoire entre les particules émises par la désintégration de la particule P lors d'un boost suivant l'axe des abscisses

On remarque qu'on a à nouveau un angle α limite, qui augmente avec la valeur de l'énergie de boost. De plus, quand la direction de P_1 est confondue avec celle de la particule P, l'angle α vaut encore π, on n'a toujours pas de retournement de la particule P_2.

4.3. Etude de la désintégration d'un Boson Z^0

Disposant désormais de connaissances basiques en relativité restreinte, en langage C++, et sachant comment utiliser les librairies Root, j'ai enfin pu commencer à étudier une « véritable » réaction : la désintégration d'un boson Z^0 en deux muons.

4.3.1. Etude simple

À partir de données Root issues de l'expérience D0 et fournies par mon directeur de stage, j'ai essayé de retrouver la masse du Boson Z^0 initial. En sommant l'énergie des muons obtenus à chaque événement ainsi que leurs impulsions, on obtient le quadrivecteur énergie-impulsion du boson Z^0. On peut ensuite simplement calculer la masse de la particule de la façon suivante :

$$E^2 = p^2c^2 + m_0^2c^4$$

En utilisant la convention $c = \hbar = 1$

$$m_0 = \sqrt{E^2 + p^2}$$

Sur la totalité des évènements relevés, on obtient alors le graphe suivant :

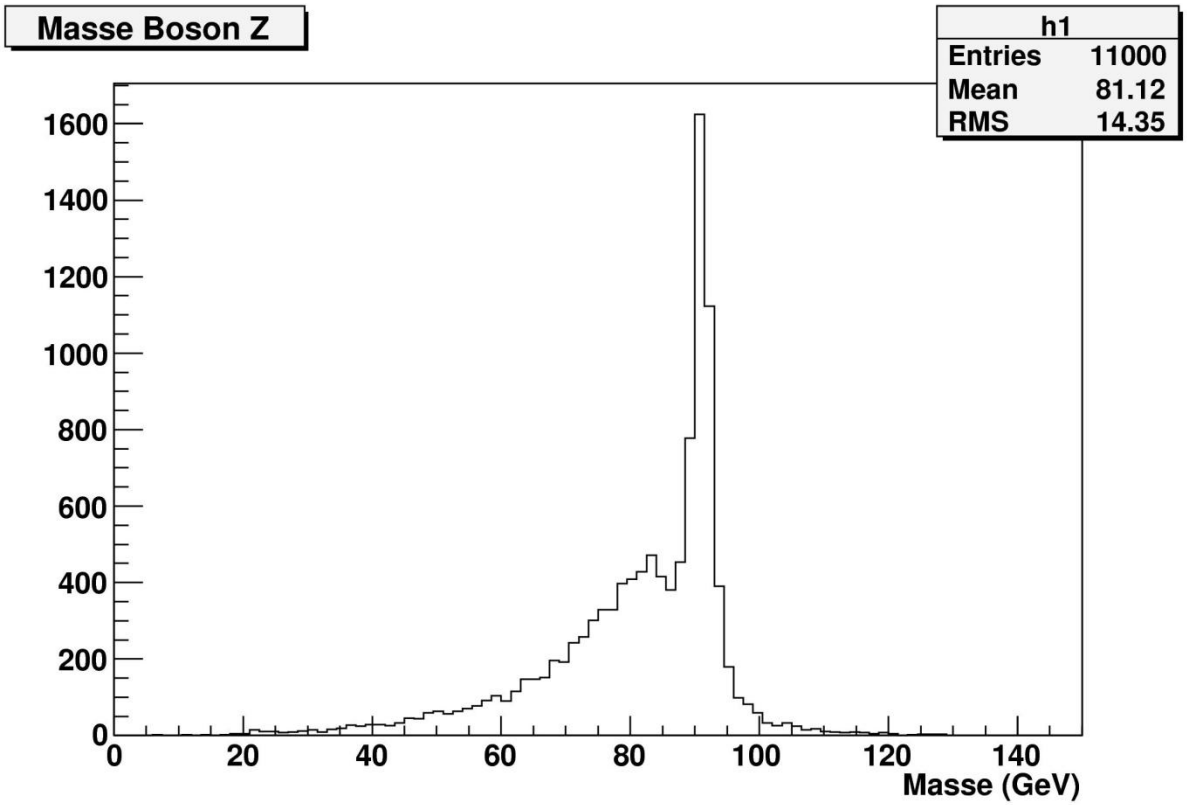

Figure 11 : Masse du Boson Z

On observe un pic autour de 90 GeV (donc proche de la véritable masse du boson Z^0). Cependant, on remarque aussi une irrégularité autour de 80 GeV. Il manque en effet une partie des données dans cette configuration, car on a juste considéré la désintégration d'un boson Z^0 en deux muons. En réalité, cette désintégration s'accompagne souvent de l'émission d'un ou plusieurs photons.

4.3.2. Correction

Cette fois-ci, j'utiliserai des données plus complètes, qui contiendront cette fois toutes les particules créées lors des évènements étudiés. Dans chaque événement contenu dans le fichier, le type de la particule est représenté par une valeur entière (13 pour le muon, 22 pour le photon). Cela permet d'être certain que les impulsions et énergies sommées sont bien celles de particules qui m'intéresse. Cette fois-ci, on obtient un graphe avec uniquement un pic autour de 91 GeV (*Figure 12 & 13).*

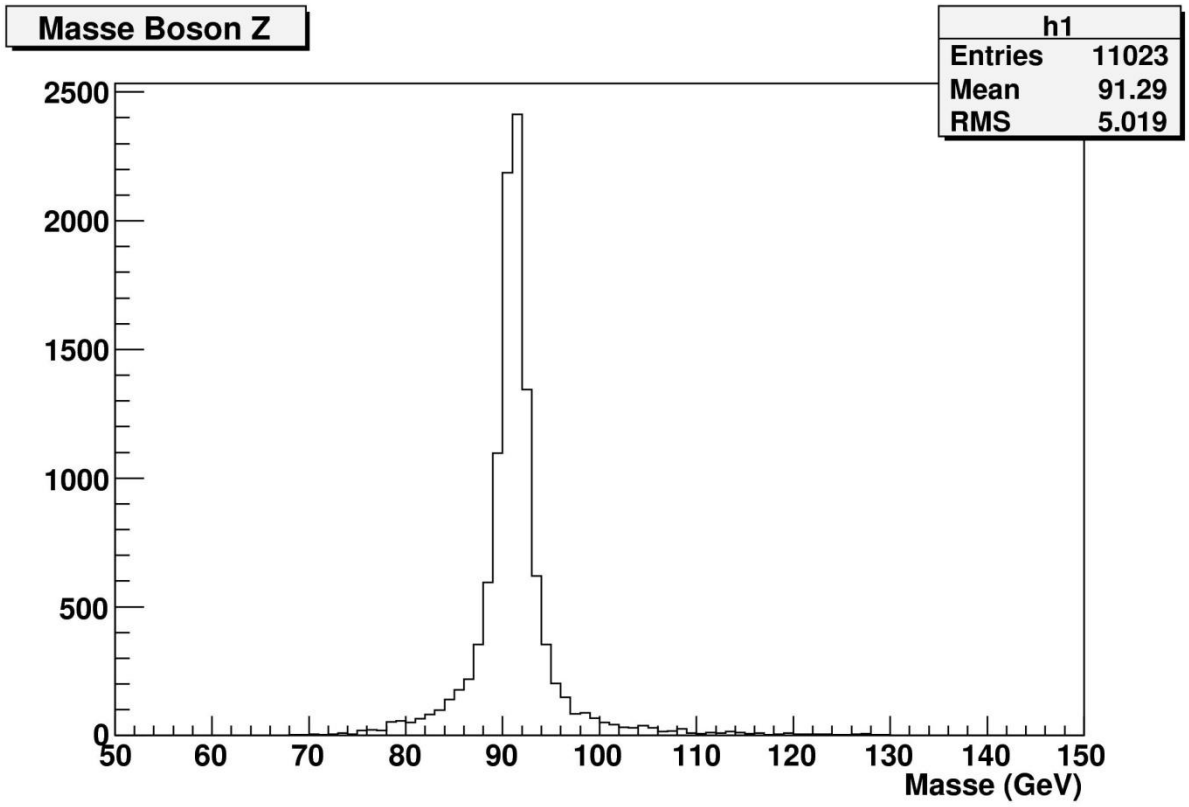

Figure 12 : Masse du Boson Z corrigée

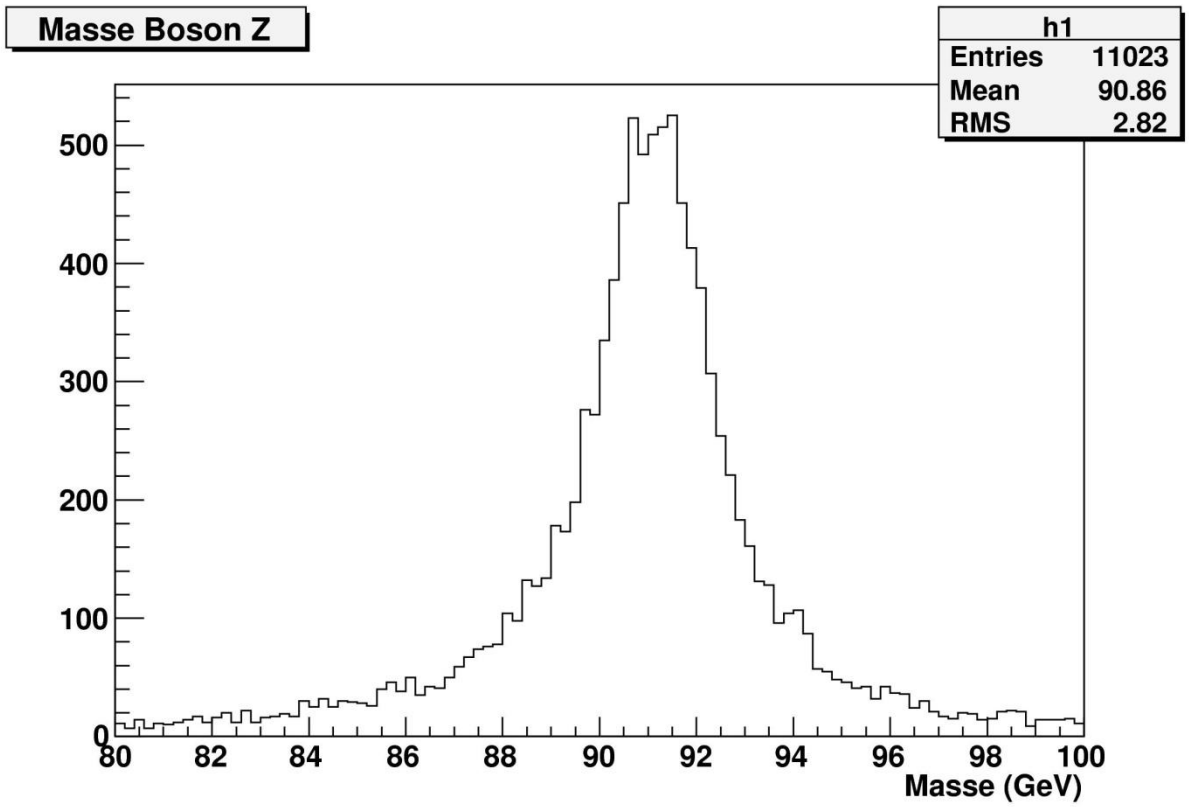

Figure 13 : Masse du Boson Z : zoom sur le pic autour de 91 GeV

La répartition de la masse du boson Z^0 semble correspondre à une courbe particulière. Dans un premier temps on essaie de lui faire correspondre une Gaussienne (*Figure 14)* d'équation :

$$f(x) \propto \frac{e^{\frac{-(x-\mu)}{2\sigma^2}}}{\sigma\sqrt{2\pi}}$$

Avec σ l'écart-type et μ l'espérance mathématique

Dans le même temps, on diminue le pas de l'histogramme : en effet, la précision des détecteurs de l'expérience D0 se limite au GeV.

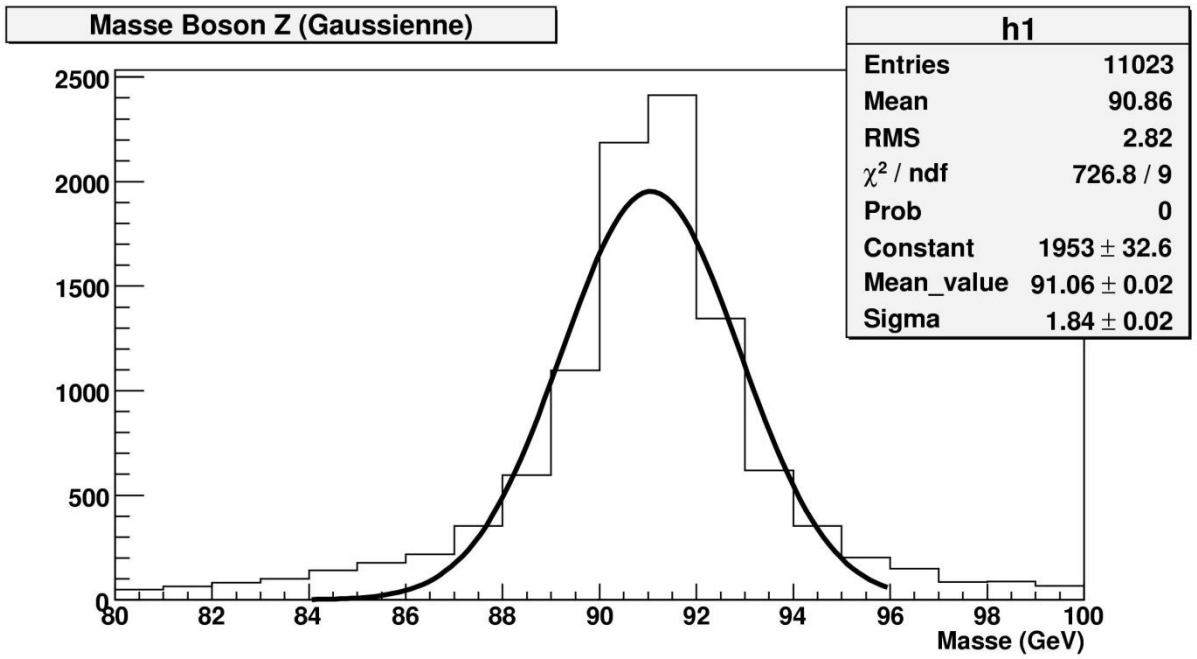

Figure 14 : Approximation par une Gaussienne

On obtient une courbe qui correspond mal aux résultats obtenus. Pour obtenir une meilleure approximation il faut plutôt utiliser une courbe de type Breit & Wigner (*Figure 15)*, d'équation :

$$f(E) \propto \frac{1}{(E^2 - M^2) + \Gamma^2 M^2}$$

Avec E l'énergie de la particule, M sa masse et Γ l'inverse de son temps de vie moyen (autrement dit, sa largeur (induite par l'inégalité de Heisenberg))

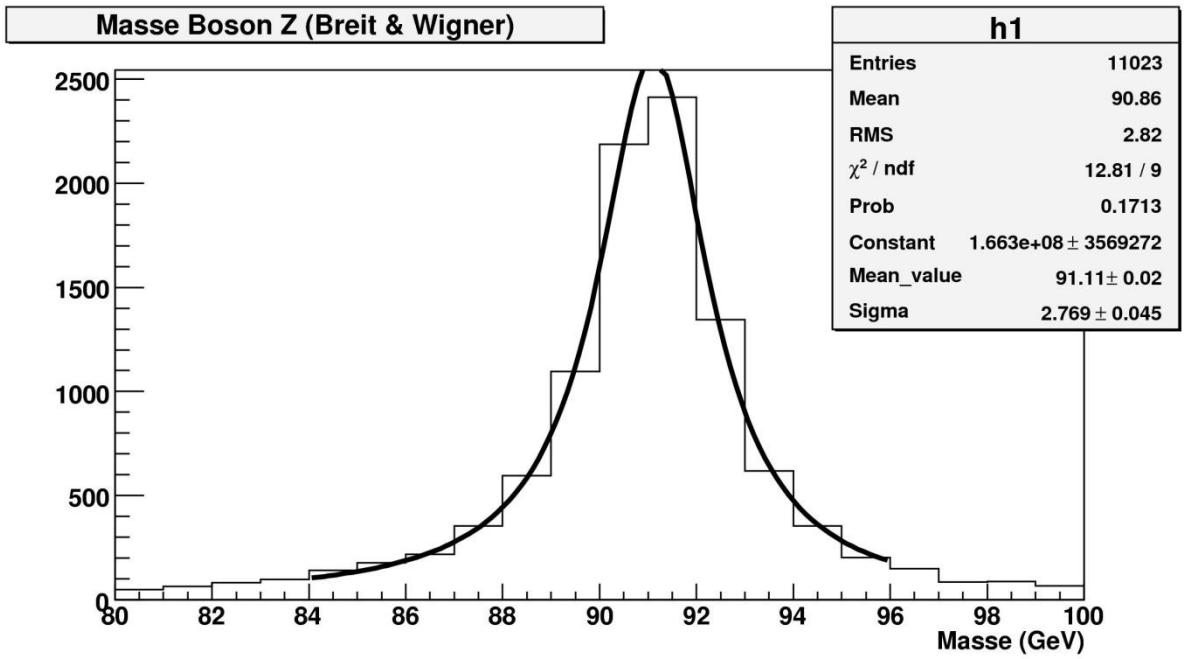

Figure 15 : Approximation par une Breit & Wigner

Le coefficient χ^2 / *ndf* (qui représente la qualité de l'approximation) est nettement plus faible avec la Breit & Wigner qu'avec la gaussienne. Il s'agit en effet de l'approximation la plus employée pour représenter la distribution en masse d'une particule élémentaire. On remarque que la valeur de la masse du Boson Z^0 observée est légèrement plus faible (environ 0.06 GeV en tenant compte des incertitudes) que la valeur officielle (91.1876 ± 0,0021 GeV [4]), mais tout de même cohérent.

5. Conclusion

Une fois les bases théoriques apprises et les outils informatiques maîtrisés, j'ai pu me lancer dans le vif du sujet : l'exploitation et le traitement de données simulant la désintégration d'un boson Z^0 en deux muons avec émission de photons. Le résultat obtenu (la masse du Boson Z^0 lors de cette réaction) est cohérente avec les observations de la communauté scientifique.

Ce stage Janus aura vraiment été une expérience enrichissante : il m'a permis de mieux appréhender le travail d'un chercheur, en levant certaines idées reçues et en me permettant de manipuler directement certains outils employés (notamment l'outil informatique, extrêmement important en physique des particules). De plus, j'ai pu approfondir mes connaissances générales notamment celles concernant les théories modernes comme le modèle standard. J'ai aussi beaucoup appris sur le fonctionnement général d'un collisionneur de particule comme le Tevatron et de ses détecteurs.

Je tiens à remercier Hélène Fonvieille de m'avoir donné la chance de pouvoir effectuer ce stage, ainsi que Philippe Gris, pour sa patience, ses conseils et sa disponibilité.

Bibliographie

[1] SEMAY C. et SILVESTRE-BRAC B. Relativité Restreinte : bases et applications. DUNOD.

[2] BATON J.P. et COHEN-TANNOUDJI G. Particules élémentaires et interactions fondamentales. DAPNIA/SPP 92-12.

[3] OUALLINE S. Practical C++ Programming. O'REILLY.

[4] YAO W.-M. {*et al.*} Review of Particle Physics, Journal of Physics G : Nuclear and Particle Physics - Volume 33 (July 2006) , Particle Data Group.

[5] *http://www.fnal.gov/*

[6] *http://www-d0.fnal.gov/*

[7] *http://root.cern.ch/drupal/*

Annexes

1. Boost C++ (classes personnelles)

- Fonction principale *boostmain.cpp* :

```
#include <iostream.h>
#include <math.h>
#include "Boost.h"


Boost::Boost(float gamma, float beta, const Q_vector& vec) : m_gamma(gamma),
m_beta(beta), m_vec(vec), m_vec_lab(vec), m_vec_lab2(vec)
{
}



void Boost::TBoost(float theta)
{

  m_vec_lab.Set_pz(m_gamma * ((m_vec.Get_P()*cos(theta)) + m_beta *
m_vec.Get_E()));
  m_vec_lab.Set_E(m_gamma * (m_vec.Get_E() + m_beta *
(m_vec.Get_P()*cos(theta))));
  m_vec_lab.Set_px(m_vec.Get_px()) ;
  m_vec_lab.Set_py(m_vec.Get_py()) ;

  m_vec_lab2.Set_pz(m_gamma * ((m_vec.Get_P()*cos(M_PI-theta)) + m_beta *
m_vec.Get_E()));
  m_vec_lab2.Set_E(m_gamma * (m_vec.Get_E() + m_beta * (m_vec.Get_P()*cos(M_PI-
theta))));
  m_vec_lab2.Set_px(m_vec.Get_px()) ;
  m_vec_lab2.Set_py(m_vec.Get_py()) ;

  cout << "\n1\n" << "px' = " << m_vec_lab2.Get_px() ;
  cout << "\n py' = " << m_vec_lab2.Get_py() ;
  cout << "\n pz' = " << m_vec_lab2.Get_pz() ;
  cout << "\n E' = " << m_vec_lab2.Get_E() ;

}
```

- Classe Q_vector :

- Q_vector.cpp :

```
#include <iostream.h>
#include <math.h>
#include "Q_vector.h"

Q_vector::Q_vector() : m_px(0), m_py(0), m_pz(0), m_E(0)
{
}


Q_vector::Q_vector(float px, float py, float pz, float E) : m_px(px), m_py(py),
m_pz(pz), m_E(E)
{
  m_P = sqrt(pow(m_px,2) + pow(m_py,2) + pow(m_pz,2));
}



Q_vector::Q_vector(const Q_vector& orig) : m_px(orig.m_px), m_py(orig.m_py),
m_pz(orig.m_pz), m_E(orig.m_E), m_P(orig.m_P)
{

}


Q_vector::~Q_vector()
{
}
```

- Q_vector.h

```
#ifndef Q_vector_h
#define Q_vector_h


class Q_vector
{
 public:
  Q_vector();
  ~Q_vector();
  Q_vector(float px, float py, float pz, float E);
  Q_vector(const Q_vector& orig);
  float Get_px();
  float Get_py();
  float Get_pz();
  float Get_E();
  float Get_P();
  void Set_px(float px);
  void Set_py(float py);
  void Set_pz(float pz);
  void Set_E(float E);
  void Set_P(float P);
```

```
 private:
  float m_px;
  float m_py;
  float m_pz;
  float m_E;
  float m_P;

  };

inline float Q_vector::Get_P() { return m_P; }
inline float Q_vector::Get_px() { return m_px; }
inline float Q_vector::Get_py() { return m_py; }
inline float Q_vector::Get_pz() { return m_pz; }
inline float Q_vector::Get_E() { return m_E; }
inline void Q_vector::Set_P(float P) { m_P = P; }
inline void Q_vector::Set_px(float px) { m_px = px; }
inline void Q_vector::Set_py(float py) { m_py = py; }
inline void Q_vector::Set_pz(float pz) { m_pz = pz; }
inline void Q_vector::Set_E(float E) { m_E = E; }



#endif
```

- Classe Boost :

- Boost.cpp

```
#include <iostream.h>
#include <math.h>
#include "Boost.h"


Boost::Boost(float gamma, float beta, const Q_vector& vec) :
m_gamma(gamma), m_beta(beta), m_vec(vec), m_vec_lab(vec), m_vec_lab2(vec)
{
}



void Boost::TBoost(float theta)
{

  m_vec_lab.Set_pz(m_gamma * ((m_vec.Get_P()*cos(theta)) + m_beta *
m_vec.Get_E()));
  m_vec_lab.Set_E(m_gamma * (m_vec.Get_E() + m_beta *
(m_vec.Get_P()*cos(theta))));
  m_vec_lab.Set_px(m_vec.Get_px()) ;
  m_vec_lab.Set_py(m_vec.Get_py()) ;

  m_vec_lab2.Set_pz(m_gamma * ((m_vec.Get_P()*cos(M_PI-theta)) + m_beta *
m_vec.Get_E()));
  m_vec_lab2.Set_E(m_gamma * (m_vec.Get_E() + m_beta *
(m_vec.Get_P()*cos(M_PI-theta))));
  m_vec_lab2.Set_px(m_vec.Get_px()) ;
  m_vec_lab2.Set_py(m_vec.Get_py()) ;

  cout << "\n1\n" << "px' = " << m_vec_lab2.Get_px() ;
  cout << "\n py' = " << m_vec_lab2.Get_py() ;
  cout << "\n pz' = " << m_vec_lab2.Get_pz() ;
  cout << "\n E' = " << m_vec_lab2.Get_E() ;

}
```

- Boost.h

```
#ifndef Boost_h
#define Boost_h
#include "Q_vector.h"

class Boost
{
 public:
  Boost(float E_lab, float mass, const Q_vector& vec);
  void TBoost(float theta);
  const Q_vector& Get_vec_lab();
  const Q_vector& Get_vec_lab2();
```

```
 private:
  float m_beta;
  float m_gamma;
  Q_vector m_vec;
  Q_vector m_vec_lab;
  Q_vector m_vec_lab2;


};

inline const Q_vector& Boost::Get_vec_lab() { return m_vec_lab; }
inline const Q_vector& Boost::Get_vec_lab2() { return m_vec_lab2; }

#endif
```

2) Boost (classe Root)

- Fonction principale *boostroot.cpp*

```
#include <Riostream.h>
#include <math.h>
#include "TLorentzVector.h"
#include "TVector3.h"
#include "TNtuple.h"
#include "TFile.h"
#include "Display.h"

//Structure pour remplir le TNtuple :
struct variables
{
  float E_lab;
  float beta;
  float gamma;
  float px1;
  float py1;
  float pz1;
  float P1;
  float px2;
  float py2;
  float pz2;
  float P2;
  float theta;
  float phi;
  float thetaboost;
  float phiboost;
  float alpha;
  float px1prime;
  float py1prime;
  float pz1prime;
  float P1prime;
  float E1;
  float px2prime;
  float py2prime;
  float pz2prime;
  float P2prime;
  float E2;
  float alphabase;
  float anglebeta;
};


int main()
{
  //Déclaration des variables :

  variables var;
  float E=45, mass=90;
  int compteur=0;
  TVector3 BetaVector(10,10,10);
```

```
  //Paramètres boucles (mininum, maximum et pas) :

  //Boucle Ã©nergie :
  float E_labmin = 200, E_labmax = 1000;
  float E_labstep = 200;


  //Boucle axe du boost :
  float phiboostmin = 0, phiboostmax = 2*M_PI;
  float phibooststep = 0.2;
  float thetaboostmin = 0, thetaboostmax = M_PI;
  float thetabooststep = 0.2;

  //Boucle axe direction particule E1 :
  float phimin = 0, phimax = 2*M_PI;
  float phistep = 0.2;
  float thetamin = 0, thetamax = M_PI;
  float thetastep = 0.2;



  TFile *f = new TFile("test.root","RECREATE");
  TNtuple *ntuple = new
TNtuple("ntuple","ntuple","E_lab:Beta:Gamma:Px1:Py1:Pz1:P1:Px2:Py2:Pz2:P2:T
heta:Phi:Thetaboost:Phiboost:Alpha:Px1':Py1':Pz1':P1':E1:Px2':Py2':Pz2':P2'
:E2:Alphabase:Anglebeta");

  var.E_lab = E_labmin;

  while (var.E_lab < E_labmax)
    {
      var.gamma = var.E_lab/mass;
      var.beta = sqrt(1 - 1/(var.gamma * var.gamma));


      var.phiboost = phiboostmin;

      while (var.phiboost < phiboostmax)
      {

        var.thetaboost = thetaboostmin;

        while (var.thetaboost < thetaboostmax)
          {
            //DÃ©finition axe du boost :


            BetaVector.SetMag(var.beta);
            BetaVector.SetPhi(var.phiboost);
            BetaVector.SetTheta(var.thetaboost);


            var.phi = phimin;

            while (var.phi < phimax)
            {

              var.theta = thetamin;
```

```
              while (var.theta < thetamax)
               {
                  //Valeurs des vecteurs non-boostés (dans le référentiel
du centre de masse R') :

                  TLorentzVector vecboost(10,10,10,E) ;
                  TLorentzVector vecboost2(10,10,10,E) ;


                  vecboost.SetRho(mass/2);
                  vecboost.SetPhi(var.phi);
                  vecboost.SetTheta(var.theta);
                  vecboost2.SetPx(-vecboost.Px());
                  vecboost2.SetPy(-vecboost.Py());
                  vecboost2.SetPz(-vecboost.Pz());



                  //Vérification de que l'angle alpha vaut PI dans le
référentiel du centre de masse R' :

                  var.alphabase = vecboost.Angle(vecboost2.Vect());



                  //Boost :

                  vecboost.Boost(BetaVector.X(), BetaVector.Y(),
BetaVector.Z());
                  vecboost2.Boost(BetaVector.X(), BetaVector.Y(),
BetaVector.Z());



                  //Valeurs des vecteurs boostés (dans le référentiel du
laboratoire R) :

                  var.pz1 = vecboost.Pz();
                  var.px1 = vecboost.Px();
                  var.py1 = vecboost.Py();
                  var.P1 = sqrt(var.pz1*var.pz1 + var.px1*var.px1 +
var.py1*var.py1);
                  var.E1 = vecboost.E();

                  var.pz2 = vecboost2.Pz();
                  var.px2 = vecboost2.Px();
                  var.py2 = vecboost2.Py();
                  var.P2 = sqrt(var.pz2*var.pz2 + var.px2*var.px2 +
var.py2*var.py2);
                  var.E2 = vecboost2.E();



                  //Valeur de alpha dans le référentiel de laboratoire R' :

                  var.alpha = vecboost.Angle(vecboost2.Vect());
                  var.anglebeta = vecboost.Angle(BetaVector);


                  ntuple->Fill(&var.E_lab);
```

```
                compteur++;
                var.theta += thetastep;

              }



            var.phi += phistep;

          }


          var.thetaboost += thetabooststep;

        }

      var.phiboost += phibooststep;

    }

    var.E_lab += E_labstep;

  }

  cout << compteur << "calculs \n";

  f->Write();

  Display::Draw_NTuple(ntuple);

  cout << "\n\n\n";


  return 0;

}
```

- Classe Display :

- Display.cpp

```
#include <iostream.h>
#include "TNtuple.h"
#include "Display.h"
#include "TCanvas.h"
#include "TApplication.h"
#include "TH1F.h"
#include "TH2F.h"


void Display::Draw_NTuple(TNtuple *ntuple)
{
  TApplication app("App",0,0);

  TH1F *h1 = new TH1F("h1","Alpha (Betaboost = 0)",100,3.12,3.15);
  TH2F *h21 = new TH2F("h21","Alpha:Anglebeta",1000,0,3.2,1000,0,3.2);
  TH2F *h22 = new TH2F("h22","h22",1000,0,3.2,1000,0,3.2);
  TH2F *h23 = new TH2F("h23","h23",1000,0,3.2,1000,0,3.2);
  TH2F *h24 = new TH2F("h24","h24",1000,0,3.2,1000,0,3.2);

  ntuple->Project("h1","Alpha","Thetaboost == Theta && Phiboost == Phi");

  TCanvas *Canvas = new TCanvas();
  Canvas->Divide(2);

  // Tracé des superpositions des fonctions Alpha(P1) pour diffèrentes
énergies
  Canvas->cd(1);
  h21->SetMarkerStyle(2);
  h21->SetMarkerColor(2);
  h21->SetMarkerSize(0.1);
  h21->SetStats(0);
  ntuple->Project("h21","Alpha:Anglebeta","E_lab == 200");
  h21->Draw();
  h22->SetMarkerStyle(2);
  h22->SetMarkerColor(3);
  h22->SetMarkerSize(0.2);
  ntuple->Project("h22","Alpha:Anglebeta","E_lab == 400");
  h22->Draw("same");
  h23->SetMarkerStyle(2);
  h23->SetMarkerColor(4);
  h23->SetMarkerSize(0.3);
  ntuple->Project("h23","Alpha:Anglebeta","E_lab == 600");
  h23->Draw("same");
  h24->SetMarkerStyle(2);
  h24->SetMarkerColor(7);
  h24->SetMarkerSize(0.4);
  ntuple->Project("h24","Alpha:Anglebeta","E_lab == 800");
  h24->Draw("same");


  Canvas->cd(2);
  h1->SetStats(0);
  h1->Draw();
```

```
  app.Run();
}
```

- Display.h

```
#ifndef Display_h
#define Display_h



namespace Display
{
  void Draw_NTuple(TNtuple *ntuple);
};

#endif
```

3) Désintégration boson Z^0

- Fonction principale *mainZmumu.cpp*

```
#include <Riostream.h>

#include <math.h>
#include "TNtuple.h"
#include "TFile.h"
#include "Display.h"
#include "Particule.h"



int main()
{

  float px[5];
  float py[5];
  float pz[5];
  float E[5];
  float npart ;
  float id[5];
  float Mass;
  Particule part[5];



  TFile *f = new TFile("Zmumu_Pierre.root");
  TNtuple *ntuple = (TNtuple*)f->Get("ntuple");
  TNtuple *result = new TNtuple("result","result","Mass");

  ntuple->SetBranchAddress("id1",&id[0]);
  ntuple->SetBranchAddress("id2",&id[1]);
  ntuple->SetBranchAddress("id3",&id[2]);
  ntuple->SetBranchAddress("id4",&id[3]);
  ntuple->SetBranchAddress("id5",&id[4]);
  ntuple->SetBranchAddress("npart",&npart);

  ntuple->SetBranchAddress("Px1",&px[0]);
  ntuple->SetBranchAddress("Py1",&py[0]);
  ntuple->SetBranchAddress("Pz1",&pz[0]);
  ntuple->SetBranchAddress("E1",&E[0]);

  ntuple->SetBranchAddress("Px2",&px[1]);
  ntuple->SetBranchAddress("Py2",&py[1]);
  ntuple->SetBranchAddress("Pz2",&pz[1]);
  ntuple->SetBranchAddress("E2",&E[1]);

  ntuple->SetBranchAddress("Px3",&px[2]);
  ntuple->SetBranchAddress("Py3",&py[2]);
  ntuple->SetBranchAddress("Pz3",&pz[2]);
  ntuple->SetBranchAddress("E3",&E[2]);

  ntuple->SetBranchAddress("Px4",&px[3]);
  ntuple->SetBranchAddress("Py4",&py[3]);
```

```
  ntuple->SetBranchAddress("Pz4",&pz[3]);
  ntuple->SetBranchAddress("E4",&E[3]);

  ntuple->SetBranchAddress("Px5",&px[4]);
  ntuple->SetBranchAddress("Py5",&py[4]);
  ntuple->SetBranchAddress("Pz5",&pz[4]);
  ntuple->SetBranchAddress("E5",&E[4]);

  int nmin = 0;


  int nentries=ntuple->GetEntries();
  for (int i = nmin ; i < nentries ; ++i)
    {

      //Lecture des entrÃ©es du ntuple :
      ntuple->GetEntry(i);

      if (npart <= 5)
      {


        TLorentzVector BosonZ(0,0,0,0);
        for (int k = 0; k < (int)npart; ++k)
          {
            part[k] = Particule(px[k], py[k], pz[k], E[k], id[k]);

             if (fabs(id[k]) == 13 || fabs(id[k]) == 22)
             {

               BosonZ = BosonZ + part[k];
             }

          }



        Mass = BosonZ.M();

        result->Fill(Mass);

      }
    }



  Display::Draw_NTuple(result);


  return 0;

}
```

- Classe particule

- particule.cpp

```
#include <iostream>

#include "Particule.h"

Particule::Particule() : TLorentzVector(), m_id(0)
{
}



Particule::Particule(float px, float py, float pz, float E, float id) :
TLorentzVector(px, py, pz, E), m_id(id)
{
}



bool Particule::IsMuon()
{
      if (fabs(m_id) == 13)
      {
            return true;
      }

      else return false;
}



bool Particule::IsPhoton()
{
      if (fabs(m_id) == 22)
      {
            return true;
      }

      else return false;
}
```

- particule.h

```
#ifndef Particule_h

#define Particule_h
#include "TLorentzVector.h"

class Particule : public TLorentzVector
{

      public :
      Particule();
      Particule(float px, float py, float pz, float E, float id);
      bool IsMuon();
      bool IsPhoton();
```

```
        private :
        float m_id ;

};

#endif
```

- Classe Display

- Display.cpp

```
#include <iostream.h>
#include <math.h>
#include "TNtuple.h"
#include "Display.h"
#include "TCanvas.h"
#include "TApplication.h"
#include "TH1F.h"
#include "TF1.h"
#include "TH2F.h"


//DÃ©finition Gaussienne :
Double_t fitf(Double_t *v, Double_t *par)
{
  Double_t arg = 0;
  if (par[2] != 0) arg = (v[0] - par[1])/par[2];

  Double_t fitval = par[0]*exp(-0.5*arg*arg);
  return fitval;
}




//DÃ©finition Breit-Wigner :
Double_t fitfbw(Double_t *v, Double_t *par)
{

  Double_t fitval =par[0]/(pow((v[0]*v[0]-par[1]*par[1]),2) +
par[1]*par[1]*par[2]*par[2]);
  return fitval;
}




void Display::Draw_NTuple(TNtuple *ntuple)
{

  TApplication app("App",0,0);
  TH1F *h1 = new TH1F("h1","Masse Boson Z (Breit &
Wigner)",1000,80,100);
  ntuple->Project("h1","Mass");
   TF1 *func = new TF1("myfunc",fitfbw,84,96,3);
  func->SetParameters(600,90,1);
  func->SetParNames("Constant","Mean_value","Sigma");
```

```
  TCanvas *Canvas = new TCanvas();


  Canvas->cd(1);
  Canvas->SetFillColor(10);
  h1->Draw();
  h1->Fit("myfunc","R");




  app.Run();

}
```

- Display.h

```
#ifndef Display_h
#define Display_h


namespace Display
{
  void Draw_NTuple(TNtuple *ntuple);
};

#endif
```